(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification⁷: G06F 9/44, H04K 1/00

(21) International Application Number: PCT/US01/42341

(22) International Filing Date:
27 September 2001 (27.09.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/676,415   28 September 2000 (28.09.2000)   US

(71) Applicant: CURL CORPORATION [US/US]; 8th Floor, 400 Technology Square, Cambridge, MA 02139-3539 (US).

(72) Inventor: BARBER, Christopher, E.; Apt. 1, 42 St. John Street, Jamaica Plain, MA 02130 (US).

(74) Agents: PISANO, Nicola, A. et al.; c/o Fish & Neave, 1251 Avenue of the Americas, New York, NY 10020 (US).

(81) Designated States (national): AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States (regional): ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:
—   as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii)) for all designations
—   as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii)) for all designations

Published:
—   with international search report

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: METHODS AND APPARATUS FOR GENERATING UNIQUE IDENTIFIERS FOR SOFTWARE COMPONENTS
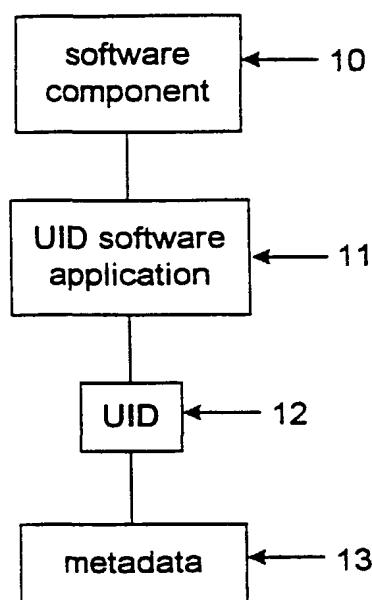
(57) Abstract: Methods and apparatus are provided for generating a unique identifier (12) for a software component (10). The unique identifier (12) is generated using a source-based, secure hash encoding. The source-based, secure hash encoding is applied to a canonical source code format of the software component. The unique identifier (12) is inserted into a meta-data structure (13) associated with the software component.

METHODS AND APPARATUS FOR GENERATING UNIQUE IDENTIFIERS
FOR SOFTWARE COMPONENTS

Field Of The Invention

The present invention relates generally to
5   methods and apparatus for generating unique identifiers
for software components. More specifically, the
present invention provides methods and apparatus for
generating unique identifiers for software components
using a source-based, secure hash.

10 Background Of The Invention

Software systems have become increasingly
more powerful and sophisticated due to the development
of component-based technology. Component-based
software systems are composed of a plurality of
15 independently deployable units of software called
components. Components generally contain software
routines or objects that perform various functions and
may be used alone or in combination with other
components. Components may also have other types of
20 content, for example, a component may contain an
interface definition, a document or a portion of a
document, data, digital media, or any other type of
independently deployable content that may be used alone
or in combination to build applications or content for
25 use on a computer. A component typically encapsulates

-2-

a collection of related data or functionality, and may
comprise one or more files.

In component-based software systems, it is
often desirable or necessary to associate a *unique*
5   *identifier*, or *UID*, with each component for locating,
registering, licensing, and communicating with or
between components. A unique identifier serves as a
unique label to designate the software component in
much the same way a license plate identifies a car, a
10   street address identifies a building, and a ISBN number
identifies a book. In all these cases, the identifier
is simply a string of characters that follows a format
designed to facilitate the interpretation of the
information the identifier is conveying. Examples of
15   such identifiers are the 9-digit ZIP code used by the
U.S. Postal Service to specify a geographical region
and the 10-digit ISBN number used to identify published
titles. Both identifiers are unique, compact, and
easily parseable.
20   Preferably, unique identifiers for software
components have many of the same properties.
Identifiers should be *unique* with respect to the
semantics of the software component and within the
domain in which they are to be used. Given the set of
25   all existing software components in a domain, no two
should have the same identifier unless the components
are semantically identical. In distributed software
applications, for example, intended to be run on
multiple machines over a network, the domain may
30   encompass all the machines that are attached to the
network. Given the size and connectivity of today's
Internet, along with its projected future growth, the
domain over which such unique identifiers may be used
could encompass a significant proportion of all of the

computers in the world. Guaranteeing the uniqueness of an identifier for a software component would significantly facilitate the design of mechanisms for locating and referencing the component among many other

5   components in a distributed network.

Identifiers should also be *compact*, that is, they should be relatively small with respect to the software components with which they are associated. Compactness allows components to be referenced quickly

10   and efficiently, and limits the space required to store the UIDs to a manageable amount. The UIDs should be as small as possible while still satisfying the uniqueness requirement.

It is also desirable that unique identifiers

15   be *secure*. Unique identifiers are secure when it is computationally infeasible to reproduce a software component corresponding to a given UID or to create two different components that produce the same identifier. Secure UIDs may also be used for authenticating

20   components to verify their origin and to verify that the components have not been modified by intruders. In this case, both the UID and the source code of the component being authenticated need to be available.

Finally, UIDs for software components should

25   be designed to be *immutable*. An UID must be a constant value that does not change as long as the software component remains the same. This guarantees that communication between components is not interrupted by a change in UIDs, and avoids broadcasting UID changes

30   to all dependent software components. If the software component itself changes significantly, the UID should then change accordingly to reflect the different functionality of the component.

There are several known mechanisms for

-4-

generating unique identifiers for software components.
The simplest approach assigns a randomly generated UID
to a software component and inserts the UID directly
into the source code of the software component. The

5   UID may be generated independently of the software
component, or it may contain a direct reference to the
component, such as using a timestamp that denotes the
time the component was created or using the name of the
software component itself.

10   Another approach involves obtaining a unique
number from a centralized location in a distributed
network and using the unique number to generate the
UID. The unique number may form part of the UID or it
may consist of the entire UID. Unique numbers need to

15 be retrieved from the centralized location whenever an
UID assignment to a software component is desired. The
retrieval process may be automatic or manual, which
would involve requesting a number from a human
administrator. Such an approach is very expensive due

20 to the often required retrievals from the centralized
location and the fact that this service may not always
be available, causing the system to generate UIDs to
fail. This was the approach used by the Cronus
distributed programming environment.

25   A UID may also be generated by combining a
timestamp with a local unique value, such as the IP
address or other network address of the host on which
the software component is being built. This approach
is used in the Component Object Model (COM)

30 specification, promulgated by Microsoft Corporation of
Redmond, WA, to assign identifiers to software
components. If there is no IP address or network
address in a given environment, a random number is used
in its place.

-5-

However, basing a UID on the location of a
software component may not be desirable. Using the IP
address combined with a timestap to form a UID reduces
the effective size of the UID, e.g., the number of bits
5   devoted to the UID, increasing the probability of
duplicating UIDs and decreasing the security of the
UID. This is due to the fact that, in general, only a
small fraction of the hosts in the network will need to
generate a UID and only a small fraction of the space
10   devoted to timestamps will be used.

Unique identifiers are usually generated by
a separate program and placed directly in the source
code of software components. However, placing UIDs
into the source code of software components does not
15   guarantee the uniqueness of UIDs. Since software
developers often reuse code, it is a real concern that
defining UIDs in the source code may cause developers
to inadvertently reuse UIDs as well. This may occur
when an identifier is accidentally copied between the
20   source code of two components or when a developer
inadvertently defines a UID that has already been
defined in another component implemented by another
developer.

An alternative approach to generating UIDs
25   has been proposed in the Java™ object serialization
specification by Sun Microsystems of Palo Alto, CA.
The UID is used as a version number for a given class
to disambiguate one version of the class from another.
Identifying the class requires the class name, which is
30   recommended to be unique by the Java™ specification,
and the UID to specify the class version. The UIDs are
declared as static variables in the source code of a
streamable class and are computed at compile time as a
64-bit *hash* of the class name, interface class name,

-6-

methods, and fields. Since the UIDs are used as
version numbers for a class, the domain over which they
operate is relatively small.

     A hash value consists of a bit field

5   generated by a *hashing algorithm* to produce a compact
representation of an input message. The Java™ object
serialization specification adopts the *Secure Hash
Standard SHA-1* as the hashing algorithm used to
generate UIDs for a streamable class. The UIDs can be

10   declared in the source code of the streamable class or
automatically generated by the Java™ runtime system and
stored in memory during execution.

     The approaches adopted by the COM
architecture and by the Java™ object serialization

15   specification to generate UIDs cannot, however,
guarantee that a UID is indeed unique. Because there
is not an inherent relationship between the identifier
and the underlying source code, it is possible for a
software component to change significantly without

20   requiring a change to its identifier as well. In the
case of Java™, for example, two software components can
contain the same class name, interface class name,
methods, and fields and have the same UIDs even though
the functions of the two components are significantly

25   different. This becomes a concern when a software
component communicates with other components and
expects the other components to provide a specific
functionality associated with their UIDs. If
significant source code changes are made to a software

30   component that causes its functionality to be
considerably different and its UID is not changed to
reflect the new functionality, it could lead to an
older piece of software using a newer version of a
software component with which it is not semantically

-7-

compatible, or vice-versa.

Additionally, the UIDs used in the Java™ object serialization specification are restricted to identifying the version number of streamable classes.
5   Large software systems may require using a UID in many other contexts, such as locating the software component, negotiating licensing information between components, and easily sharing components in a software system implemented using multiple programming
10 languages. It would be desirable to store the UID together with a variety of descriptive information about the software component so the UID of a software component can be automatically extracted together with bibliographical data, version number, licensing, and
15 other descriptive information.

In view of the drawbacks associated with previously-known methods and apparatus for generating a unique identifier for a software component, it would be desirable to provide methods and apparatus suitable for
20 generating a unique identifier for a software component that overcome these drawbacks.

It would further be desirable to provide methods and apparatus for generating an identifier for a software component that reflects the functionality of
25 the software component.

It would still further be desirable to provide methods and apparatus for generating an identifier for a software component that is unique, compact, immutable, and secure.
30   It would also be desirable to provide methods and apparatus for automatically generating a unique _ identifier for a software component and inserting the unique identifier in a *meta-data* structure associated with the software component.

Summary Of The Invention

        In view of the foregoing, it is an object of
the present invention to provide methods and apparatus
5   suitable for generating a unique identifier for a
software component that overcome drawbacks associated
with previously-known methods and apparatus.

        It is another object of the present invention
to provide methods and apparatus for generating an
10 identifier for a software component that is based on
the functionality of the software component.

        It is a further object of the present
invention to provide methods and apparatus for
generating an identifier for a software component that
15 is unique, compact, immutable, and secure.

        It is also an object of the present invention
to provide methods and apparatus for automatically
generating a unique identifier for a software component
and inserting the unique identifier in a *meta-data*
20 structure associated with the software component.

        These and other objects of the present
invention are accomplished by providing methods and
apparatus for generating a unique identifier for a
software component using a secure hash encoding of the
25 source code implementation of the software component.
The unique identifier is inserted in a meta-data
structure associated with the software component.

        In a preferred embodiment, the methods of the
present invention for automatically generating a unique
30 identifier for a software component involves three
major steps: (1) putting the source code text of the
software component into a canonical form; (2) computing
a hash encoding based on the canonical form of the
source code text to form the unique identifier; and

-9-

(3) associating the unique identifier with the software
component.

More specifically, the present invention uses
a cryptographically secure hashing algorithm as the
5   basis for generating a unique identifier for a software
component. The unique identifier is computed by
selecting the portions of the source code text that are
important to the implementation of the software
component. The selected source code text is encoded
10 into a sequence of bits using a standard character
encoding algorithm. A cryptographically secure hashing
algorithm is applied to the sequence of bits to
generate a unique, compact, immutable, and secure hash
value that is used as the unique identifier. Since the
15 cost of the secure hashing algorithm is proportional to
the number of input bits, the character encoding
algorithm should be chosen to produce the most compact
encoding for the expected source code. The unique
identifier is then associated with the software
20 component, for example, by inserting it into a meta-
data structure. Preferably, the unique identifier is
generated automatically by special purpose routines
that are executed when compiled versions of the
software component are created, and embedded in the
25 generated object code. Having the unique identifier be
generated automatically at compile time avoids having
to read the source code twice, one to compile it, and
one to generate the UID. Further, embedding the UID in
the object code of the software component ensures that
30 the UID is associated with the most up-to-date version
of the component.

An important advantage of the present
invention inheres in the fact that the unique
identifier is computed using only the portions of the

-10-

source code text that are important to the
implementation of the software component. This ensures
that insignificant changes in the source code of the
software component do not produce a different unique
5   identifier for the component. Only changes with
respect to the component's functionality would lead to
a different unique identifier. This reduces the
possibility of assigning the same identifier to two
software components that have distinct functions.
10   Another technical advantage of the present
invention inheres in the fact that the unique
identifier of a software component is automatically
generated when the source code is being compiled and
thus requires no human intervention and presents no
15   possibility of human error.
            In the preferred embodiment of the invention,
the UID is inserted into a meta-data structure
associated with the compiled software component. This
enables unique identifiers of software components to be
20   easily accessible for several purposes, including
location, registration, and licensing of components.
            Other technical advantages of the present
invention will become apparent to one of ordinary skill
in the art in view of the drawings and detailed
25   description of the embodiments of the invention.

Brief Description Of The Drawings
            The above and other objects of the present
invention will be apparent upon consideration of the
following detailed description, taken in conjunction
30   with the accompanying drawings, in which like reference
characters refer to like parts throughout, and in
which:
            FIG. 1 is a schematic view of a preferred

-11-

embodiment of a software system for generating a unique
identifier for a software component;

        FIG. 2 is a schematic view of the structure
of a meta-data associated with a software component
5   built in accordance with the principles of the present
invention;

        FIG. 3 is a simplified block diagram of an
illustrative software application for automatically
generating a unique identifier for a software component
10 using a source-based, secure hash in accordance with
the principles of the present invention;

        FIG. 4 is a flow chart for putting the source
code text of the software component into a canonical
form;

15   FIGs. 5A and 5B are schematic views of two
embodiments of a process for generating a unique
identifier for a software component;

        FIG. 6 is a schematic view of a computer
system suitable for use with the present invention; and

20   FIG. 7 is an illustrative computing
environment on which the methods and apparatus of the
present invention may be used.

Detailed Description Of The Invention

25   Referring to FIG. 1, a preferred embodiment
of a system for generating a unique identifier for a
software component is described. Software component 10
is an independently deployable unit of software that
may be used, e.g., "imported", by other software.
30 Components generally contain software routines or
objects that perform various functions and may be used
alone or in combination with other components.
Components may also have other types of content, for
example, a component may contain an interface

-12-

definition, a document or a portion of a document,
data, digital media, or any other type of independently
deployable content that may be used alone or in
combination to build applications or content for use on
5   a computer. Components preferably contain well-defined
interfaces, so that one or more components can be
interconnected to build software applications, without
requiring detailed knowledge of the inner workings of
the components. A component typically encapsulates a
10 collection of related data and functionality, and may
comprise one or more files. For example, a component
may consist of a single file that implements a bounding
box for text, a component may consist of multiple files
that implement a spell checker that uses the component
15 for building a bounding box, or a component may consist
of an entire word processor, using multiple components
that implement diverse functions such as generating
bounding boxes or checking the spelling.

        The source code text of software component 10
20 is used as an input to UID software application 11,
which comprises a set of special purpose routines for
automatically generating a UID for a software
component. UID software application 11 may be built as
a separate custom-built software application or it may
25 be integrated into programming language compilers or a
development environment. In a preferred embodiment of
the present invention, the UID software application is
part of the compiler, since the compiler must read the
source code to perform its functions.

30   UID software application 11 generates as
output unique identifier (UID) 12. As used herein, a
UID is a series of bits or a bit field that serves as a
label associated with a software component. A UID can
be used for identifying, locating, registering,

-13-

licensing, and communicating with components. In a
preferred embodiment of the invention, UID software
application 11 ensures that UID 12 is unique within the
domain it is used, compact and relatively small with
5   respect to software component 10, immutable, and
secure. Upon being generated, UID 12 is inserted into
meta-data structure 13 associated with software
component 10.

        Referring now to FIG. 2, the structure of
10   meta-data 13 according to the principles of the present
invention is described. Meta-data 13 includes a
variety of information about software component 10.
This information should minimally include the name of
software component 10 and UID 12. Alternatively, UID
15   12 may be used as the only reference mechanism for
software component 10, which in this case, would not
necessarily have a name. Meta-data 13 may also include
a variety of other information, as described in detail
hereinbelow. Parts of meta-data 13 may be specified by
20   a programmer at the time software component 10 is
written, while other parts must be computed, and added
to meta-data 13. UID 12 is an example of a part of
meta-data 13 that is computed by UID software
application 11. Similarly, meta-data such as the time
25   at which a component was released would typically be
computed and added to meta-data 13 by a release tool or
by a development environment.

        As shown in FIG. 2, meta-data 13 comprises a
series of key-value pairs. Keys 14a - 14d identify
30   particular items of meta-data by name. For example,
unique identifier 12 may use "id" as a key, while a
meta-data item containing the name of the author of a
component may use "author" as a key. Values 15a - 15d
are meta-data values associated with keys 14a - 14d.

-14-

These values contain information pertaining to the
particular component with which the meta-data is
associated.

   In a preferred embodiment, meta-data 13 may
5   include the items listed in table 1. It will be
understood that the meta-data associated with a
component may vary, and that some systems will use a
different set of meta-data than that shown in FIG. 2 or
table 1. Additionally, it will be understood that some
10   of the meta-data items listed in table 1, such as the
name of the component, may be stored separately from
other key-value pairs, due to their presence in almost
all components, or the frequency with which they are
accessed. Certain of the meta-data items listed in
15   Table 1, such as "author", and "name", will be given
values by the programmer or content developer, while
others, such as "compilation-time" and "id" will be
generated by a compiler or development environment.

TABLE 1 - Common Meta-data Items	
Meta-data Key	Description
name	The name of the component.
version	The version number of the component.
id	Unique ID for the component.
author	The author(s) of the component.
compilation-time	The time at which the instance of the component was compiled.
component-type	The type (e.g., "package" or "document") of the component.
copyright	A copyright notice for the component.

-15-

files	A list of the files included in the component.
frozen?	Indicates whether the component is "frozen". If the component is "frozen", any attempt to change its meta-data will result in an error.
jit-compiled?	Indicates whether the component instance was JIT compiled.
source-url-name	The URL of the file from which the component was compiled.
system-versions	The versions of the system on which the component may be used.
transitive-id	A unique component ID that incorporates the IDs of all components on which the component depends.

Referring to FIG. 3, a block diagram of UID software application 11 is described.

UID software application 11 consists of three stages. First, at stage 16, a canonical source code format is generated by selecting only the portions of the source code text that are important to the implementation of the software component. Second, at stage 17, the canonical form of the source code text is encoded into a sequence of bits using a character encoding algorithm. Lastly, at stage 18, a hashing algorithm is applied to the sequence of bits to generate a hash value that is used as the UID associated with a software component.

Referring now to FIG. 4, a flow chart of stage 16 for generating a canonical source code format used to represent the software component is described.

-16-

The canonical source code format is generated to avoid creating a different identifier every time an insignificant change is made on the source code that does not affect the semantics or implementation of the

5   software component. A source code change is insignificant when it does not affect the semantics or implementation of the software component. The canonical source code format removes all the insignificant portions of the source code text. The

10   insignificant portions of source code text depend on the programming language used to write the source code text. Examples of insignificant portions of source code text include certain whitespaces, certain comments, and other characters.

15   At step 20, any character that is not significant to the semantics or implementation of the software component is discarded. Unless it would alter the functionality of the software component, any remaining whitespaces are converted to a canonical

20   whitespace character, typically the space character. Significant whitespaces are those required by the programming language specification or those that have an effect on the output of the software component. Examples of significant whitespaces include spaces

25   between the type definition of a variable and its name, and spaces included in strings that are part of the output of the software component. Insignificant whitespaces include spaces used for indentation of the source code and spaces in the source code comments.

30   For example, consider the following fragment of code implemented using the C programming language:

```
int
main()
{
```

-17-

```
        printf("Hello      World!");
        return 0;
    }
```

After removing insignificant characters at step 20, the
5   code above is reduced to:
```
        int main(){printf("Hello      World!");return 0;}
```


The above example shows that the extra spaces in the
string "Hello   World!" are not discarded because
this would affect the output of the software component.
10   If the source code text is not case sensitive
(step 21), then the selected source code text is
converted at step 22 to a single case, either lower
case or upper case. At step 23, all comments included
by a developer that are not significant to the output
15 and functionality of the software component are
discarded. Special documentation comments that are
supported in certain programming languages are not
discarded, since they form an important part of the
software component.
20   At step 24, a decision is made on whether the
source code used to represent the software component is
selected solely from portions that denote the external
interface of the software component. The external
interface of a software component is used to specify
25 how the software component interacts or communicates
with other software components. The software component
may be represented solely by its external interface
when, for example, the software component is
implemented by multiple programming languages. In case
30 the source code used to represent the software
component is to denote only the external interface,
then those parts of the source code that do not
contribute to this external interface are omitted at

step 25.

          Referring back to FIG. 3, upon generating the
canonical source code format at stage 16, stage 17
encodes the canonical source code format into a
5   sequence of bits. The sequence of bits is generated
using a standard character encoding algorithm, such as
ASCII, ISO Latin-1, UCS-2, or UTF-8, or a custom-built
character encoding algorithm. The character encoding
algorithm selected should support all the characters
10   that occur in the programming language used to write
the source code text and produce the most compact
sequence of bits for the canonical source code format.
For example, the portion "int main" of the canonical
source code format described above
15   int main(){printf("Hello   World!");return 0;}
may be encoded into the following sequence of bits
using the 7-bit ASCII character encoding:
          1101001 1101110 1110100 0100000 1101101 1100001
          1101001 1101110
20   Each character of the "int main" string is converted
into a 7-bit field. The UTF-8 algorithm may be a good
choice when the underlying source code accepts the
Unicode character set but it is expected that most
characters in the source code are in the ASCII set.
25   The sequence of bits is then used at stage 18
by a *hashing algorithm* to generate a hash value that
forms the UID for the software component. A hash value
consists of a bit field that is generated by a hashing
algorithm to produce a compact representation of an
30   input message. For example, the character encoding
described above for the string "int main"
          1101001 1101110 1110100 0100000 1101101 1100001
          1101001 1101110
may be reduced to the following hash value by using a

-19-

simple hashing algorithm that consists of combining all
of the bytes of the string using bitwise exclusive-or:
        1011000
Because the hash values are in general much smaller
5   than the original message, there can be more than one
message that will produce the same hash value; this is
known as a *hash collision*. Such collisions are
unacceptable if the hash encoding is intended to be
used as a unique identifier. Two messages should have
10   the same hash value if and only if the messages are
identical. The hash encoding algorithm should
therefore be chosen to provide an extremely low
probability of collisions over the set of expected
messages to be encoded. For practical implementation
15   purposes, the hashing algorithm should also be chosen
such that the hash value is easily computable on a
message of arbitrary length.
        One group of hashing algorithms that fits
this description is the set of cryptographically secure
20   hashing algorithms, which have been designed for use in
data authentication and encryption. Such algorithms
are designed to produce a secure digest of a given
message that can be used to authenticate the contents
of the message. Examples of such algorithms include
25   MD-5, RIPEMD-128, RIPEMD-160, and the *Secure Hash
Standard SHA-1* adopted by all Federal departments and
agencies for the protection of unclassified
information. In a preferred embodiment of the
invention, the SHA-1 algorithm is used to generate the
30   UID for a software component because it generates 160-
bit hash values instead of the 128-bit hash values
generated by RIPEMD-128 and MD-5. The main advantage
is that the larger number of hash values generated with
the 160-bit algorithms than with the 128-bit algorithms

-20-

provides a lower probability of collisions. Although
RIPEMD-160 also generates 160-bit hash values, the SHA-
1 algorithm is more widely used in the U.S. and is
recommended by the U.S. government.

5   Referring now to FIGS. 5A and 5B, two
illustrative embodiments of a process for generating a
unique identifier for a multi-file software component
are described. Component source codes 28 and 29
contain the source code of different files of software

10   component 27. The files may represent parts of source
code text from component 27 such as header files or
library files, or they may represent individual
components or interfaces that are used in combination
to form component 27. Component source codes 28 and 29

15   may be written in a variety of languages, including
programming languages such as C and Java™, markup
languages such as HTML, hybrid content/programming
languages such as CURL (described in commonly owned,
copending, U.S. patent application No. _____

20   (CURL-002)), and interface definition languages (IDLs)
such as CORBA IDL. In multi-language environments, in
particular, in environments involving languages that
have closely related syntax, it may be necessary to
encode the name and version of the language in the UID

25   to distinguish between software components with
identical source code that behave differently because
the components were compiled using different language
rules.

          With respect to FIG. 5A, software component

30   source codes 28 and 29 are combined by concatenation
routine 30 to generate a single piece of source code
that is used by UID software application 11 to form UID
31. The order in which the components are concatenated
may be the alphabetical order of software component

-21-

source codes 28 and 29, the order in which software
component source codes 28 and 29 are compiled, the
order according to the number of lines in software
component source codes 28 and 29, or an order

5   established by a sorting algorithm that sorts software
component source codes 28 and 29 according to their
contents. Any method to determine the order should be
used consistently, so that the same set of source code
files generates the same UID regardless of the order in

10 which the files are processed.

UID software application 11 generates UID 31
based on the single piece of source code formed at 30
by concatenating software component source codes 28 and
29. UID 31 is then inserted in a meta-data structure

15 32 associated with software component 27.

With respect to FIG. 5B, UID software
application 11 is applied individually on software
component source codes 28 and 29 to generate a *file
identifier* UID for each of software component source

20 codes 28 and 29. A file identifier is a UID that
identifies a source code file of a software component.
UID 33 is generated for software component source code
28 and UID 34 is generated for software component
source code 29. UIDs 33 and 34 are combined at 35 to

25 form UID 36. It is important to combine the UIDs using
operations that generate a UID that is unique, compact,
immutable, and secure. One requirement is that the
domain of the operations not be significantly smaller
than the domain of the unique identifiers; that is,

30 given an arbitrary identifier *I*, the set of identifiers
produced by applying the operations to I and every
possible identifier is the set of all possible
identifiers, or very close to it.

One set of operations that fits this

-22-

description are the operations which form an algebraic
group over the set of possible identifiers. It is
assumed that the set of possible identifiers is the
same as the set of integers less than a particular
5   power of two, which is referred to as n. Three
operations may be used to combine the UIDs: exclusive-
or, 2's complement addition, and addition modulo 2^n.
The exclusive-or and 2's complement addition operations
are the fastest, and should preferably be used in
10   practice. The exclusive-or operation should be avoided
in case there is a significant probability of having to
combine UIDs that are identical. This occurs for
example, when there are identical pieces of source code
text. In this case, applying the exclusive-or
15   operation to identical UIDs would produce a zero-value
UID. An example includes a software component that
contains backups of multiple pieces of source code text
that are identical to the original pieces of source
code.

20   If software component source codes 28 and 29
contain the source code of individual components or
interfaces that are used in combination to form
software component 27, then the UID 36 is called a
transitive UID. The transitive ID is a unique
25   component ID that incorporates the IDs of all
components on which component 27 depends.

            It will be understood by one skilled in the
art that other combination methods could also be used
in accordance with the principles of the present
30   invention, and that any of these techniques may be used
alone, or in combination.

            Referring now to FIG. 6, a computer capable
of generating a UID for a software component in
accordance with the principles of the present invention

-23-

is described. Computer system 38 includes at least
processor 39, for processing information according to
programmed instructions for generating a UID for a
software component, memory 40, for storing information
5   and instructions for processor 39, and storage system
41, such as a magnetic or optical disk system, for
storing large amounts of information and instructions
on a relatively long-term basis. Processor 39, memory
40, and storage system 41 are coupled to bus 42, which
10   preferably provides a high-speed means for devices
connected to bus 42 to communicate with each other.

It will be apparent to one of ordinary skill
in the art that computer system 38 is illustrative, and
that alternative systems and architectures may be used
15   with the present invention. It will further be
understood that many other devices, such as a display
system (not shown), a network interface (not shown),
and a variety of other input and output devices (not
shown) may be included in computer system 38.

20   Referring to FIG. 7, an example of a
computing environment in which the methods and
apparatus of the present invention may be used is
described. Computers 43, 44, and 45, and server 46 are
connected to local area network (LAN) 47. Each of
25   computers 43, 44, and 45 may execute a variety of
software, all or part of which may be stored locally on
computers 43, 44, or 45, or may be stored on server 46,
and accessed over LAN 47.

LAN 47 is connected to a wide area network
30   (WAN) 48, such as the Internet, through gateway 49,
which may be a dedicated device, or may be a computer
or server, similar to computers 43, 44, and 45, or
server 46. By sending communications across WAN 48,
any of the devices connected to LAN 47 may communicate

-24-

with remote server 50, or with other devices or networks connected to WAN 48. Computers 43, 44, and 45 may gain access to information and software components on other computers through WAN 48.

5   It will be understood by one skilled in the art that the above-described environment is for illustration, and that the present invention may be used under a variety of conditions. For example, the methods and apparatus of the present invention could be

10   used on a single stand-alone computer, but would then be unable to access information or software over LAN 47 or WAN 48. Generally, the present invention may be used with any configuration of one or more computers, which may be interconnected on one or more networks.

15   Although particular embodiments of the present invention have been described above in detail, it will be understood that this description is merely for purposes of illustration. Specific features of the invention are shown in some drawings and not in others,

20   and this is for convenience only and any feature may be combined with another in accordance with the invention. Steps of the described processes may be reordered or combined, and other steps may be included. Further variations will be apparent to one skilled in the art

25   in light of this disclosure and are intended to fall within the scope of the appended claims.

-25-

What Is Claimed Is:

      1.   A method of generating a unique identifier for a software component, the method comprising:

      transforming source code text of the software component into a canonical source code format;

      encoding the canonical source code format into a sequence of bits;

      computing a hash of the sequence of bits to form the unique identifier; and

      associating the unique identifier with the software component.

      2.   The method of claim 1, wherein associating the identifier with the software component comprises inserting the identifier in a meta-data structure.

      3.   The method of claim 1, wherein the software component comprises a plurality of source code files, and wherein transforming source code text of the software component into a canonical source code format involves concatenating the source code files into a single file.

      4.   The method of claim 3, wherein concatenating the source code files into a single file comprises ordering the source code files.

      5.   The method of claim 4, wherein ordering the source code files comprises selecting an order from a group consisting of:

      alphabetical order of the name of the source code files;

order according to the number of lines in the source code files;

order in which the source code files are compiled; and

order established by a sorting algorithm that sorts the source code files according to their contents.

6.   The method of claim 1, wherein transforming source code text of the software component into a canonical source code format comprises

discarding insignificant characters;

converting remaining whitespaces into a canonical whitespace character if the remaining whitespaces do not have an effect on the compiled component;

determining whether the programming language used to write the source code text is case-insensitive, and if so, converting the respective source code text into a single case; and

discarding insignificant comments.

7.   The method of claim 6, further comprising discarding all portions of the source text that do not contribute to an external interface of the software component if the software component is to be identified only by its external interfaces.

8..   The method of claim 1, wherein encoding the canonical source code format into a sequence of bits comprises encoding the canonical source code format using a character encoding algorithm.

-27-

9.   The method of claim 1, wherein computing
a hash of the sequence of bits to form the unique
identifier comprises applying a cryptographically
secure hashing algorithm on the sequence of bits.

10.   The method of claim 1, wherein the meta-
data structure comprises a variety of descriptive
information about the software component, such as
bibliographical information, version number of the
software component, compilation information, and
licensing information.

11.   A method of generating a unique
identifier for a software component consisting of a
plurality of source code files, the method comprising:
        generating a unique file identifier for each
source code file;
        combining the file identifiers to form the
unique identifier for the software component; and·
        inserting the unique identifier into a meta-
data structure associated with the software component.

12.   The method of claim 11, wherein
generating a unique file identifier for a source code
file comprises
        transforming the source code text of the
source code file into a canonical source code format;
        encoding the canonical source code format
into a sequence of bits;
        computing a hash of the sequence of bits to
form the file identifier.

-28-

13.   The method of claim 12, wherein
transforming source code text of the source code file
into a canonical source code format comprises
         discarding insignificant characters;
         converting remaining whitespaces into a
canonical whitespace character if the remaining
whitespaces do not have an effect on the compiled
component;
         determining whether the programming language
used to write the source code text is case-insensitive,
and if so, converting the respective source code text
into a single case; and
         discarding insignificant comments.

14.   The method of claim 13, further
comprising discarding all portions of the source text
that do not contribute to external interfaces of the
source code file if the file identifier for the source
code file is to denote only the external interfaces of
the source code file.

15.   The method of claim 12, wherein encoding
the canonical source code format into a sequence of
bits comprises encoding the canonical source code
format using a character encoding algorithm.

16.   The method of claim 12, wherein
computing a hash of the sequence of bits to form the
file identifier comprises applying a cryptographically
secure hashing algorithm on the sequence of bits.

-29-

17.   The method of claim 11, wherein combining the file identifiers to form the unique identifier comprises using a bitwise exclusive-or routine.

18.   The method of claim 11, wherein combining the file identifiers to form the unique identifier comprises using an addition modulo 2^n routine, wherein n is the number of bits used to represent the file identifiers.

19.   The method of claim 11, wherein combining the file identifiers to form the unique identifier comprises using an addition modulo p routine, wherein p is the largest prime number smaller than 2^n, wherein n is the number of bits used to represent the file identifiers.

20.   The method of claim 11, wherein the meta-data structure comprises a variety of descriptive information about the software component, such as bibliographical information, version number of the software component, compilation information, and licensing information.

21.   A software component, comprising a unique identifier that represents a secure hash of the significant portions of source code text of the software component, and a meta-data structure in which the unique identifier is stored along with a variety of descriptive information about the software component.

22.   The software component of claim 21,
wherein a secure hash of the significant portions of
source code text of the software component comprises
applying a cryptographically secure hashing algorithm
to the significant portions of source code text of the
software component.

23.   The software component of claim 22,
wherein the significant portions of source code text of
the software component comprise the portions of source
code text that are important to the implementation of
the software component.

24.   The software component of claim 23,
wherein the portions of source code text that are
important to the implementation of the software
component comprise source code text that determines the
functionality of the software component and certain
source code comments.

25.   The software component of claim 21,
wherein the meta-data structure comprises a variety of
descriptive information about the software component,
such as bibliographical information, version number of
the software component, compilation information, and
licensing information.

26.   An apparatus for generating unique
identifiers for a software component, the apparatus
comprising:
          a general purpose computer;
          a memory, the memory storing a plurality of
routines for execution by the processor, wherein the
plurality of routines comprise:

-31-

a transformation routine that selects source code text from the software component to generate a canonical source code format;

an encoding routine to encode the canonical source code format into a sequence of bits;

a hash encoding routine to compute a hash value from the sequence of bits to form the unique identifier;

numerical operation routines to combine multiple hash values into a single hash value; and

an embedding routine to insert the unique identifier into a meta-data structure associated with the software component.

27. The apparatus of claim 26, wherein the transformation routine comprises the steps of:

a) discarding insignificant whitespaces and converting any remaining whitespaces into a canonical whitespace character if the remaining whitespaces do not have an effect on the compiled component;

b) determining whether the programming languages used to implement the software components are case-insensitive, and if so, transforming respective source code into a single case; and

c) discarding insignificant comments.

28. The apparatus of claim 27, further comprising discarding all portions of the source code text that do not contribute to external interfaces of the software component if the software component is to be identified only by its external interface.

-32-

29.   The apparatus of claim 26, wherein the
encoding routine comprises applying a character
encoding algorithm to the canonical source code format.

30.   The apparatus of claim 26, wherein the
hash encoding routine comprises generating a hash value
using a cryptographically secure hashing algorithm for
the sequence of bits.

31.   The apparatus of claim 26, wherein
numerical operation routines to combine multiple hash
values into a single hash value comprises:
        a bitwise exclusive-or routine;
        a routine that performs addition modulo 2^n,
wherein n is the number of bits used to represent the
hash values; and
        a routine that performs addition modulo p,
wherein p is the largest prime number smaller than 2^n.

32.   The apparatus of claim 26, wherein the
meta-data structure comprises a variety of descriptive
information about the software component, such as
bibliographical information, version number of the
software component, compilation information, and
licensing information.

FIG. 1

FIG. 2

FIG. 3

FIG. 4

FIG. 5A

FIG. 5B

38

40

39 → | Processor | | Memory | | Storage System | ← 41

42

FIG. 6

48 → Internet

46 →

49

50 →

47

43   44   45
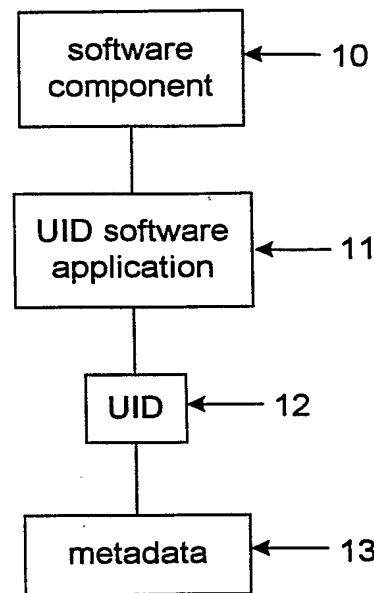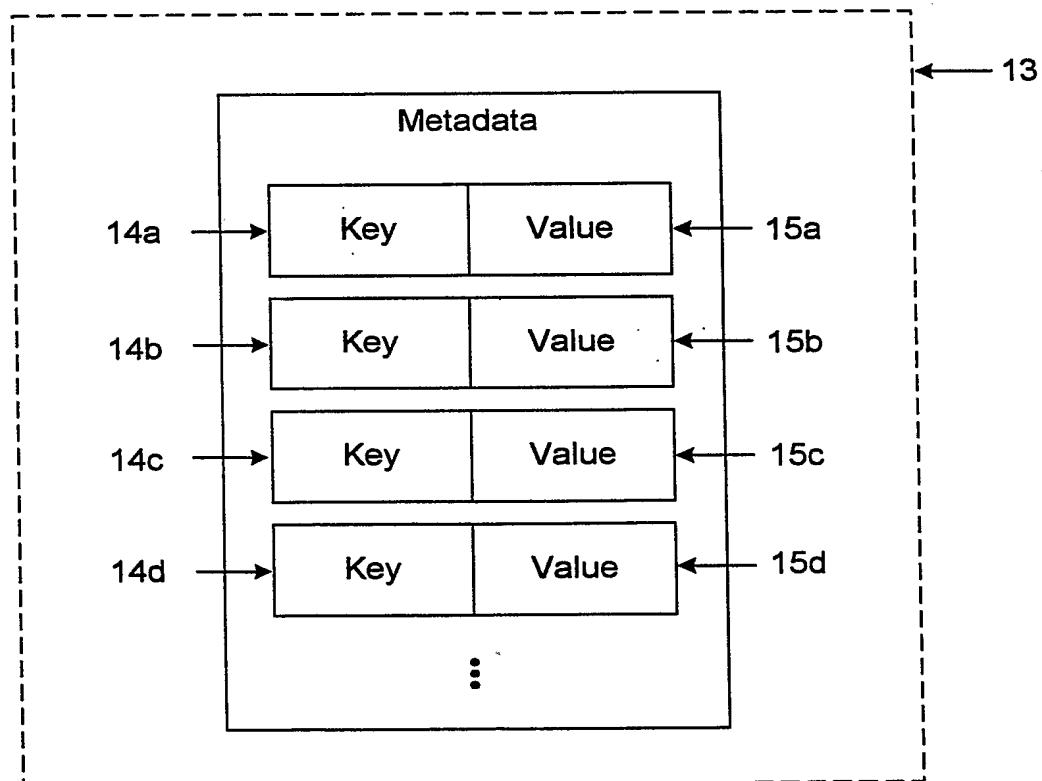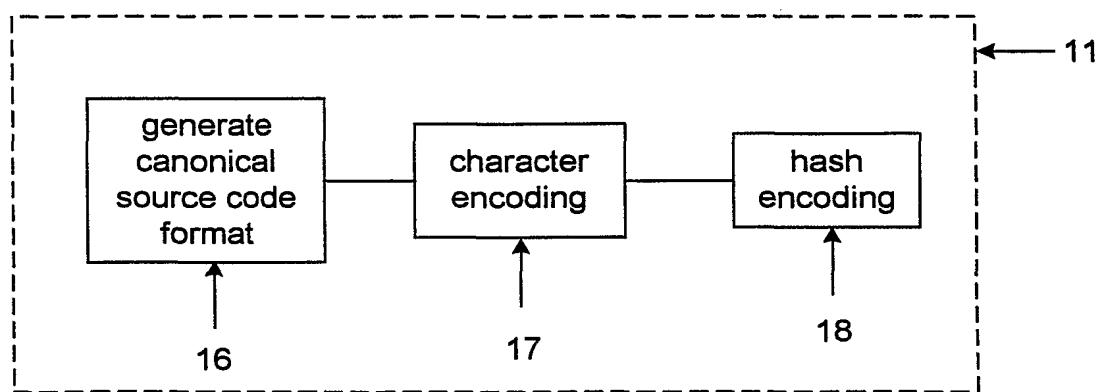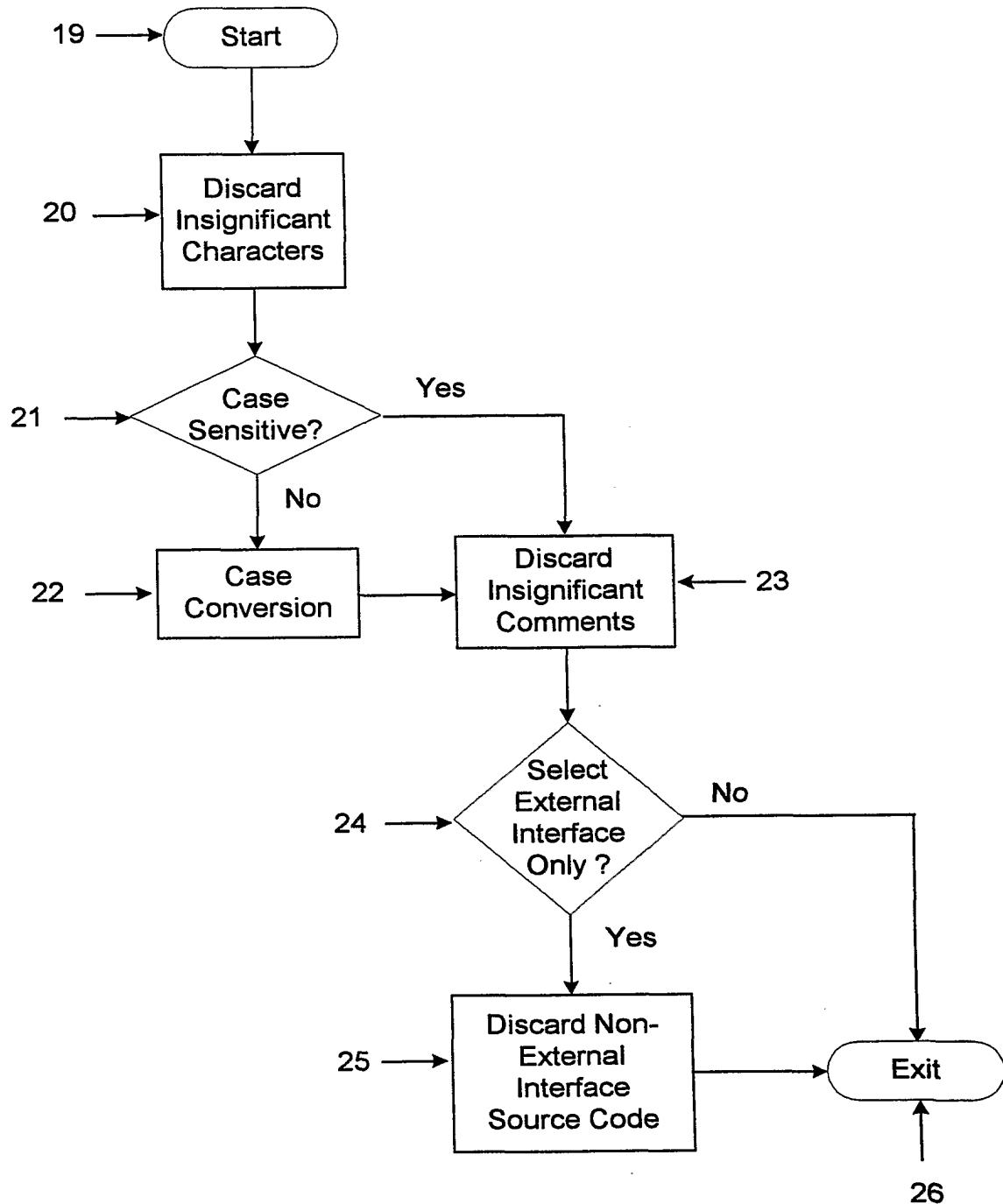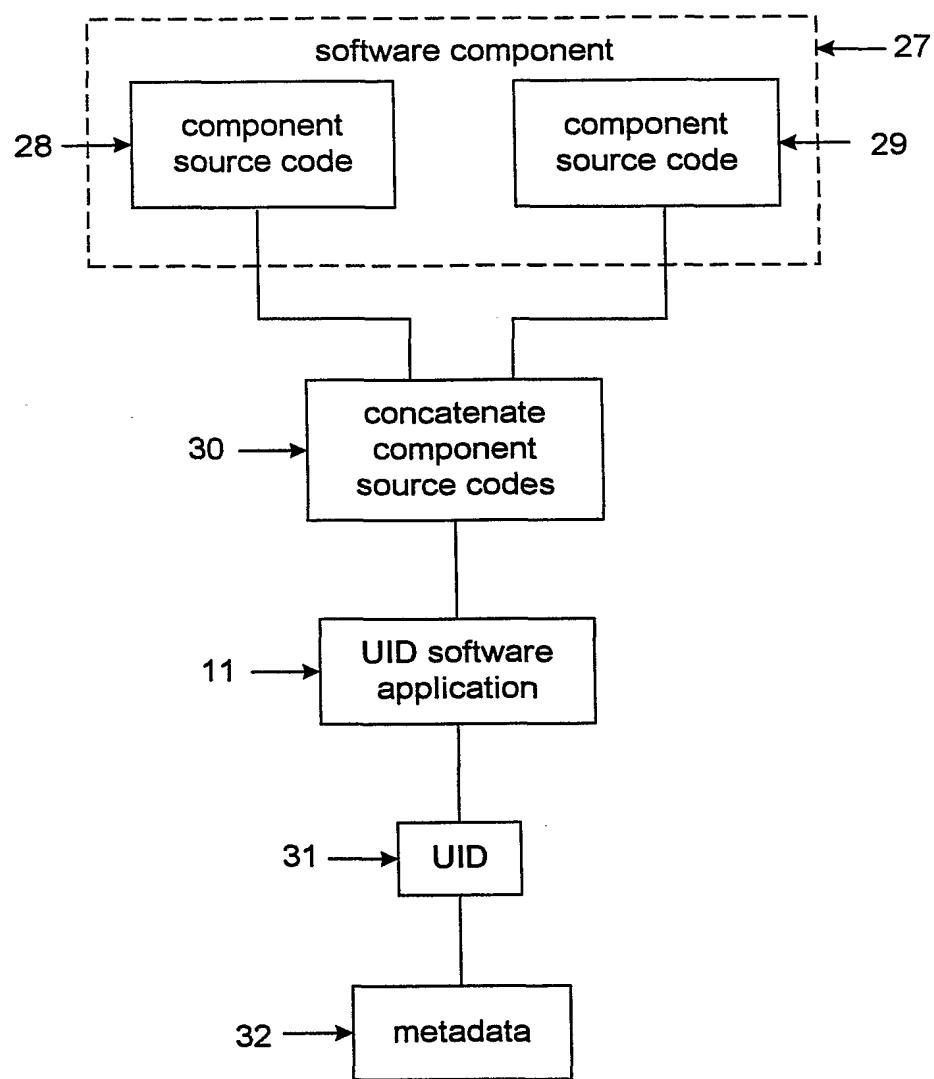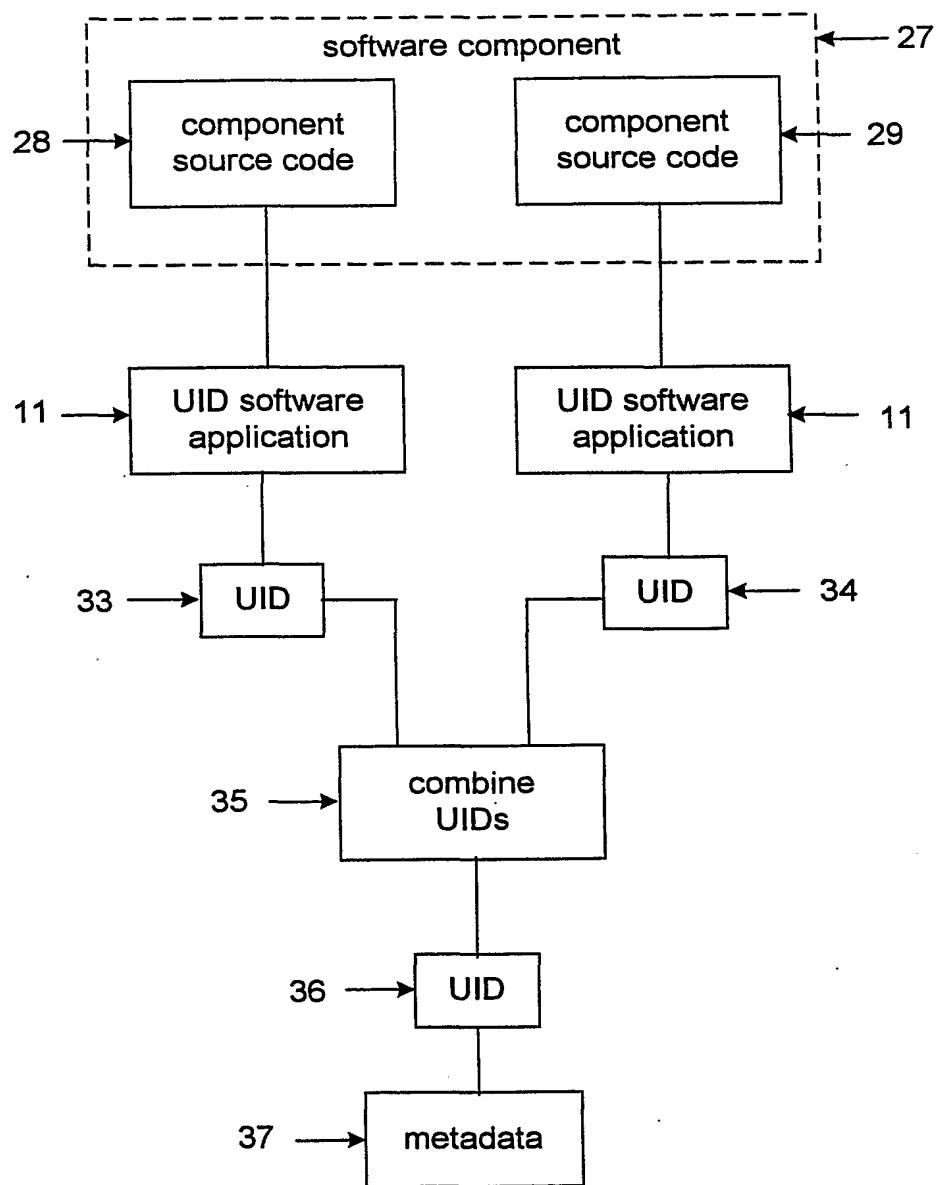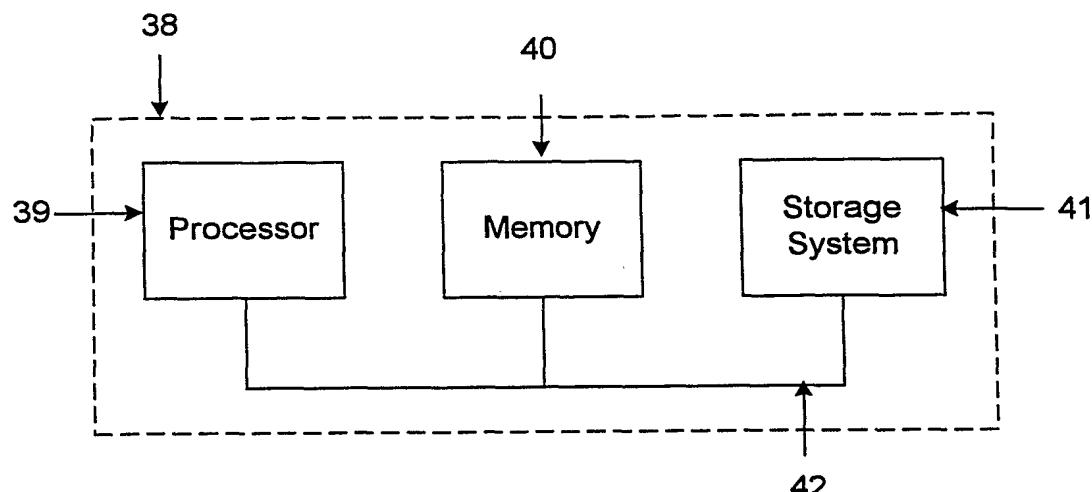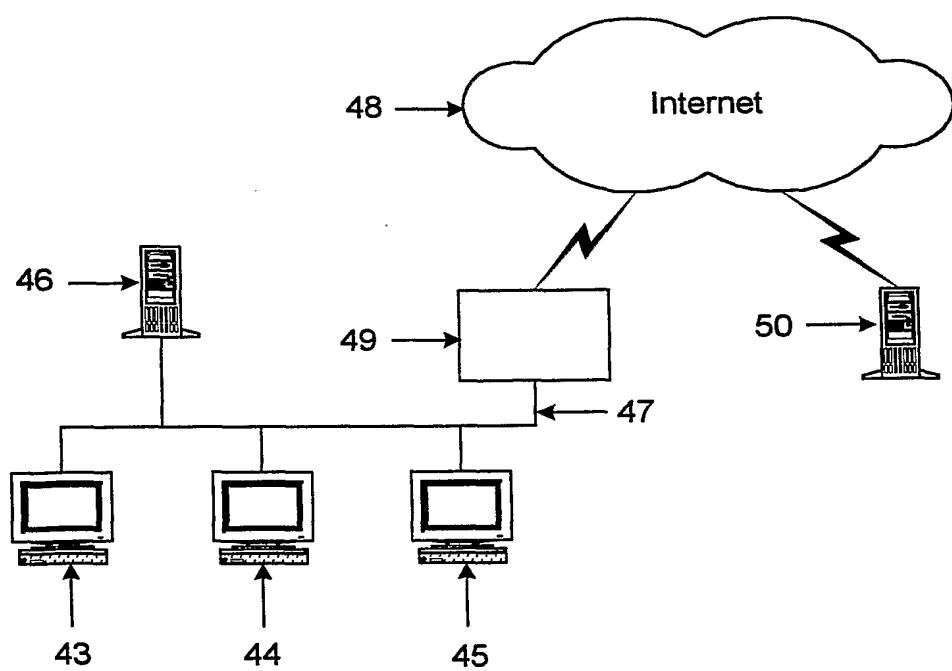
FIG. 7

INTERNATIONAL SEARCH REPORT

A. CLASSIFICATION OF SUBJECT MATTER

IPC(7)   :G06F 9/44; H04K 1/00
US CL   :380/44; 713/167; 717/10, 11
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. :   380/44; 713/167; 717/10, 11

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
  WEST

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 5,500,881 A (LEVIN et al) 19 March 1996, col. 3, line 14 to col. 4, line 14; col. 8, line 65 to col. 10, line 44; col. 12, line 57 to col. col. 18, line 19; col. 18, line 29 to col. 55, line 67.	1-32
Y	US 6,098,054 A (MCCOLLOM et al) 01 August 2000, col. 3, line 66 to col. 8, line 28.	1-32
Y,P	US 6,243,468 B1 (PEARCE et al) 05 June 2001, col. 3, line 66 to col. 9, line 14.	1-32

☐ Further documents are listed in the continuation of Box C.   ☐   See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance		
"E"	earlier document published on or after the international filing date	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed	"&"	document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
17 NOVEMBER 2001	**17 DEC 2001**

Name and mailing address of the ISA/US	Authorized officer
Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231	MATTHEW SMITHERS
Facsimile No. (703) 305-3230	Telephone No. (703) 308-9293

Form PCT/ISA/210 (second sheet) (July 1998)*